

# COGS109: Lecture 10



Regression, interpolation and extrapolation introduced

July 20, 2023

***Modeling and Data Analysis***

Summer Session 1, 2023

C. Alex Simpkins Jr., Ph.D.

RDPRobotics LLC | Dept. of CogSci, UCSD

# Models and the modeling process



# Linear least squares

- You're probably all familiar with linear regression -- fitting a line to a bunch of data.
- more formally fitting  $y = mx + b$  for paired  $x, y$  data (can also do multidimensional)
- Let's see how it's done mathematically

## Let's start by considering an easier question...

- We have 2 points, and want to fit a line to them
- $(1,2)$  ,  $(3,4)$
- How would you solve this problem?
- We want  $y=mx+b$  (we need **m** and **b**)
  - **Substitute each point in**

$$2 = m(1) + b$$

$$4 = m(3) + b$$

# Example continued

- And solve for **b** first, then **m**

$$b = 2 - m$$

$$4 = 3m + 2 - m$$

$$4 = 2m + 2$$

$$m = 1$$

$$b = 2 - m$$

$$b = 1$$

## Example continued

- We have two equations and two unknowns ( $m, b$ )
- This can be written compactly as

$$\begin{bmatrix} 1 & 1 \\ 3 & 1 \end{bmatrix} \begin{Bmatrix} m \\ b \end{Bmatrix} = \begin{Bmatrix} 2 \\ 4 \end{Bmatrix}$$

- Which is of the basic form

$$Ax = b$$

- We want to find

$$x = A^{-1}b$$

# Solving $Ax=b$

- Solving for  $x = A^{-1}b$  involves computing the inverse of the A matrix
  - **Insiwhatsitz? Don't worry...inverses are a way to make life easier**
- There are several methods, and you can solve for arbitrarily sized problems (ie what if we want to find 100 variables? Not fun by hand: ( Let's use a computer to do it for us!!!:))
  - **Gaussian elimination (what you learned in linear algebra class)**
    - Don't worry you won't have to do it by hand in this class!
  - **Thomas algorithm, etc (and other more efficient methods computationally)**
  - **Python has gaussian elimination (and others) built-in nicely of course through numpy modules**

# We need to remind ourselves of matrix inversion

- What is an inverse of a matrix?
- Rotation example
  - **If a vector is rotated by multiplying it by a rotation matrix, then multiplying the rotated vector by the inverse rotates the vector back to its original orientation**
  - **Side note - a matrix times its inverse yields the identity matrix**
    - You can test for a matrix being the inverse of another matrix by multiplying the two, and see how close do you get to the identity matrix?

$$\boxed{AA^{-1} = I} \quad \boxed{A^{-1}A = I} \quad \boxed{AI = A} \quad \boxed{IA = A}$$

- **Look up more of the definition details...see references on site**
    - Homework problem, one matrix plot is an example...which could it be? Hmm...what special matrices have we just mentioned? Hmmm...how could I IDENTIFY this matrix? Hmmm...
- Dating example



# Solving $Ax=b$

- We compute the solution of our canonical problem by

$$A^{-1}A = I$$

$$Ix = x$$

*Recall*

*that...*

$$Ax = b$$

$$A^{-1}Ax = A^{-1}b$$

$$Ix = A^{-1}b$$


$$x = A^{-1}b$$

# How to solve $Ax=b$ in matlab

- In python/numpy this can be solved for with the “numpy.linalg.lstsq” method
- Returns the least squares solution to a linear matrix equation
- <https://numpy.org/doc/stable/reference/generated/numpy.linalg.lstsq.html>
  - **roughly the same as  $INV(A)*B$**
  - **computed in a different way.**
    - If A is an N-by-N matrix and B is a column vector with N components, or a matrix with several such columns, then  $A = np.linalg.lstsq(X,B)$  is the solution to the equation  $A*X = B$  computed by a method that depends on the input matrices.
- Doing it in python:

(<https://numpy.org/doc/stable/reference/generated/numpy.linalg.lstsq.html>)

Custom colormaps and pcolor

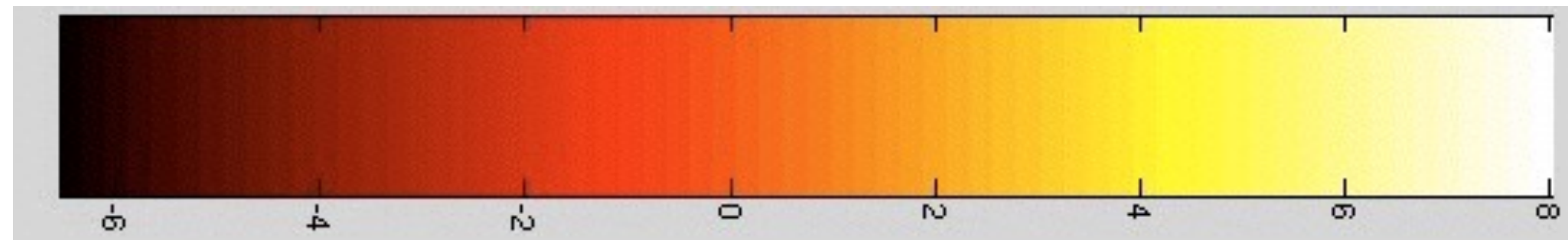


# Outline for this section

- Colormap implementation
- What is interpolation?
  - **Definition**
  - **Applications, motivation for use**
  - **Orion nebula simulation**
- LERP - Linear interpolation
- BERP - Bilinear interpolation
- TERP - Trilinear interpolation
- SLERP - Spherical linear interpolation in polar coordinates
- Examples

# Creating color maps - review and expansion

- What if I want to examine the boundaries of my data?
  - **I only want to see the extremes**
  - **We can create a custom color map!**



# Creating the color map (r,g,b) components

- To create a custom color map we need to make a matrix which is Dim  $n \times 3$ , range  $[0,1]$
- Each column is the range of either red, green, blue
- Writing it by hand:

$$M = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

- Typing it into a python variable:  
**M = ?**



# Now what?

- We create our plot, let's create some data and plot it using *pcolor*:
  - [https://matplotlib.org/stable/api/\\_as\\_gen/matplotlib.pyplot.pcolor.html](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.pcolor.html)
  - [https://matplotlib.org/stable/gallery/images\\_contours\\_and\\_fields/pcolor\\_demo.html#sphx-glr-gallery-images-contours-and-fields-pcolor-demo-py](https://matplotlib.org/stable/gallery/images_contours_and_fields/pcolor_demo.html#sphx-glr-gallery-images-contours-and-fields-pcolor-demo-py)
- Matlab
  - `peaks(50)`
  - `pcolor(X)`
  - `colormap(M)`

# Here's what we get...

- As you can see this can be very useful for feature detection
- But let's say we want to make a smooth map, how do we do that?







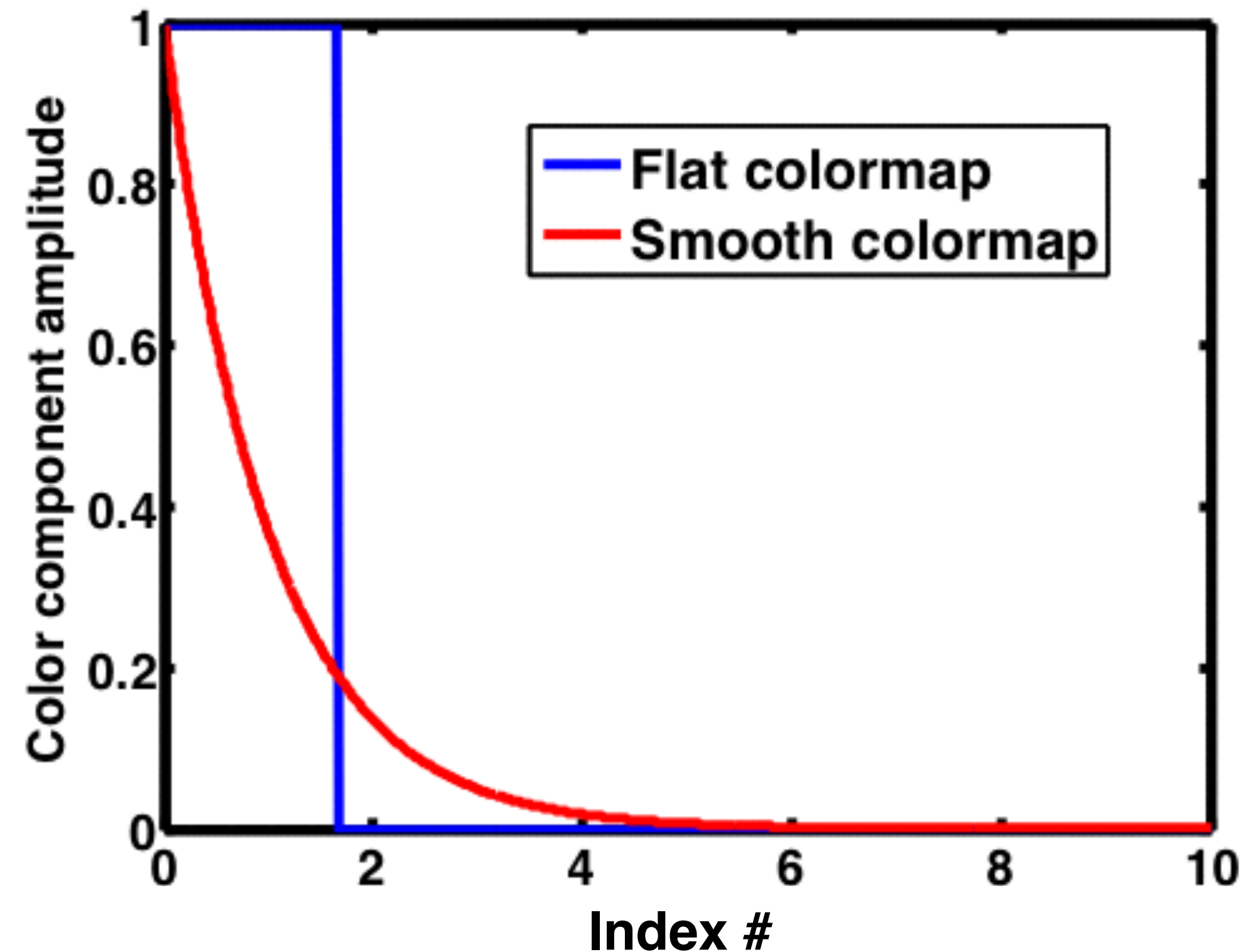
# Creating smooth color map functions

- Instead of typing the matrix in manually, let's construct the functions we need to make transitions smooth from one color to the next
- Create many values in between 0 and 1
- Two things of note
  - **The length of your colormap array is up to you, the more numbers and the smaller the transitions, the more smooth the colors look (crayons vs. airbrushing)**
  - **The colors are mapped so the**  
**range(0,1) -> range( min(data), max(data) )**

# Looking at smooth transitions

- Comparison after matching the number of values in the simple color variation (1 -> 0) vs. a smooth function from 1->0
- Uses the equation...
  - **(for Decreasing:)**

$$r = \exp(-x)$$
$$x = 0 : .01 : 10$$



# The final smooth color map

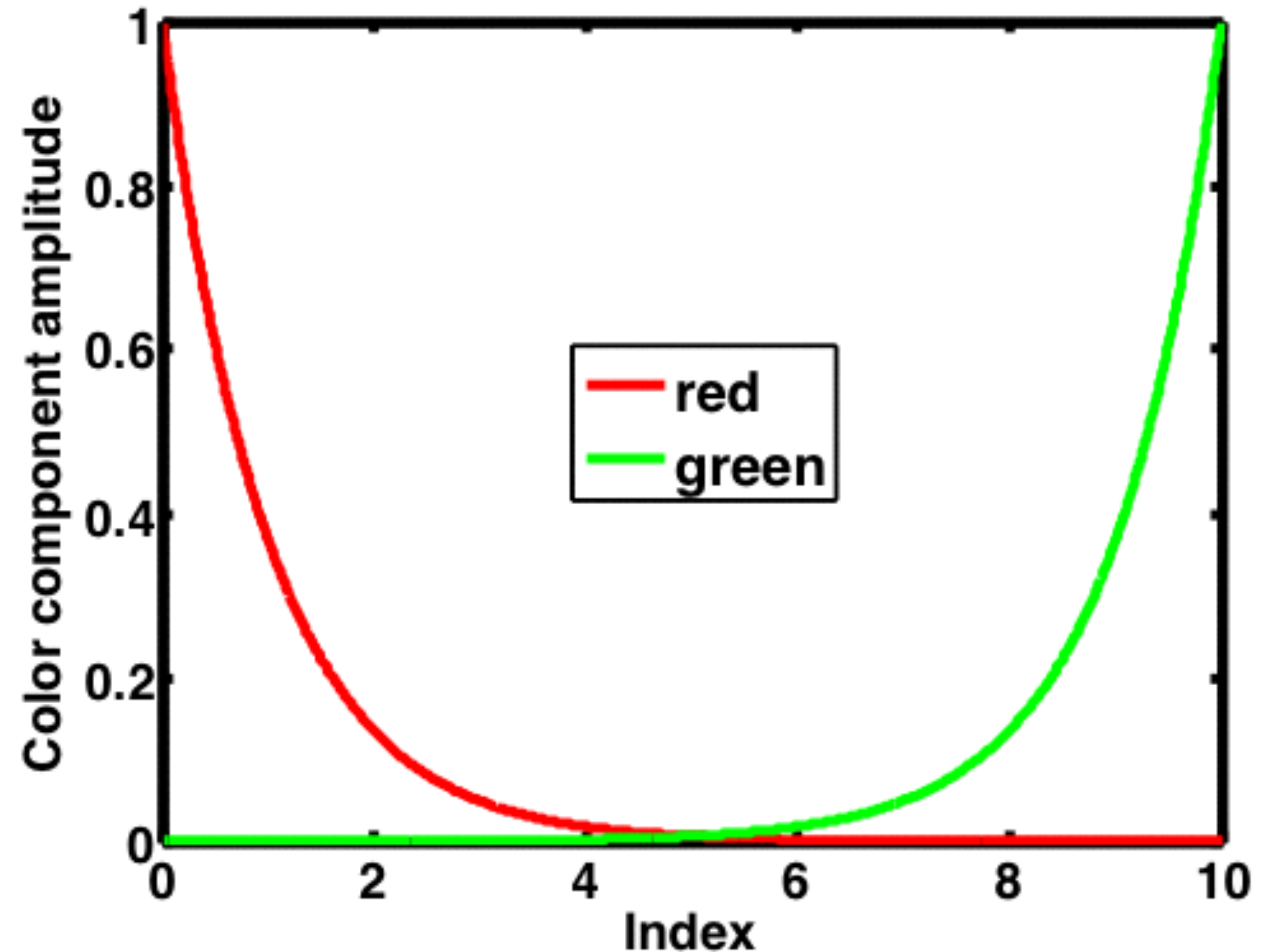
- And equations:

- **Decreasing:**

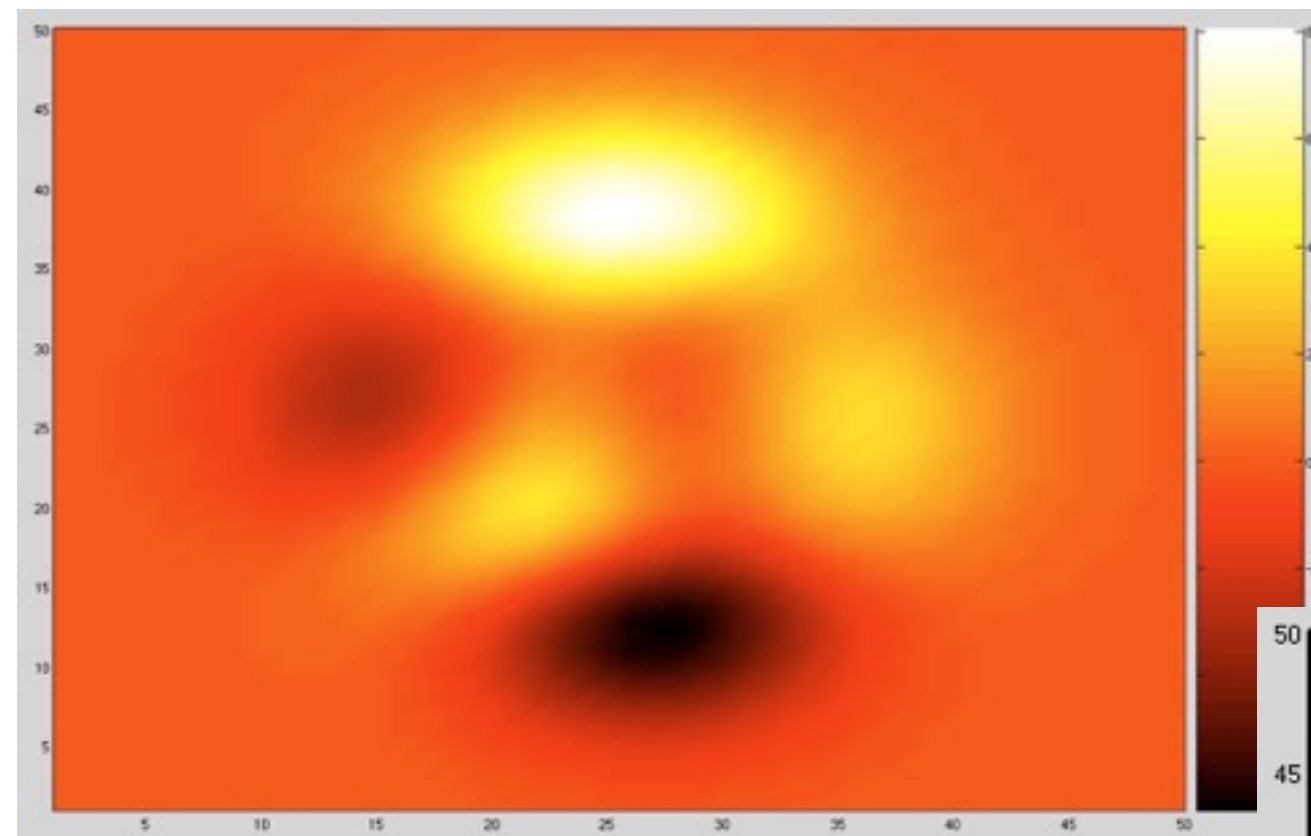
$$r = \exp(-x)$$
$$x = 0 : .01 : 10$$

- **Increasing:**

$$g = \frac{\exp(x)}{\max[\exp(x)]}$$
$$x = 0 : .01 : 10$$

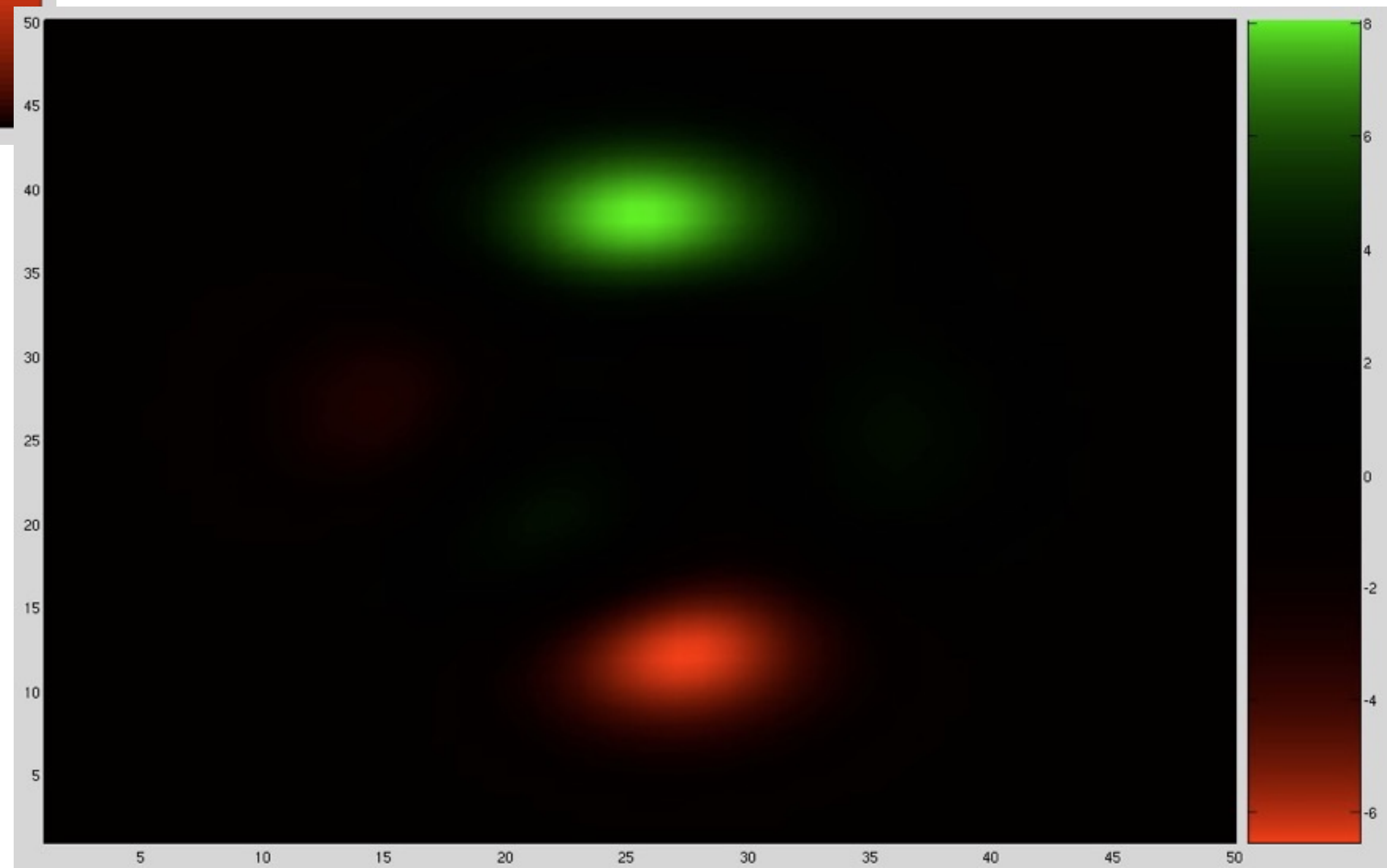


# Results of our custom color map



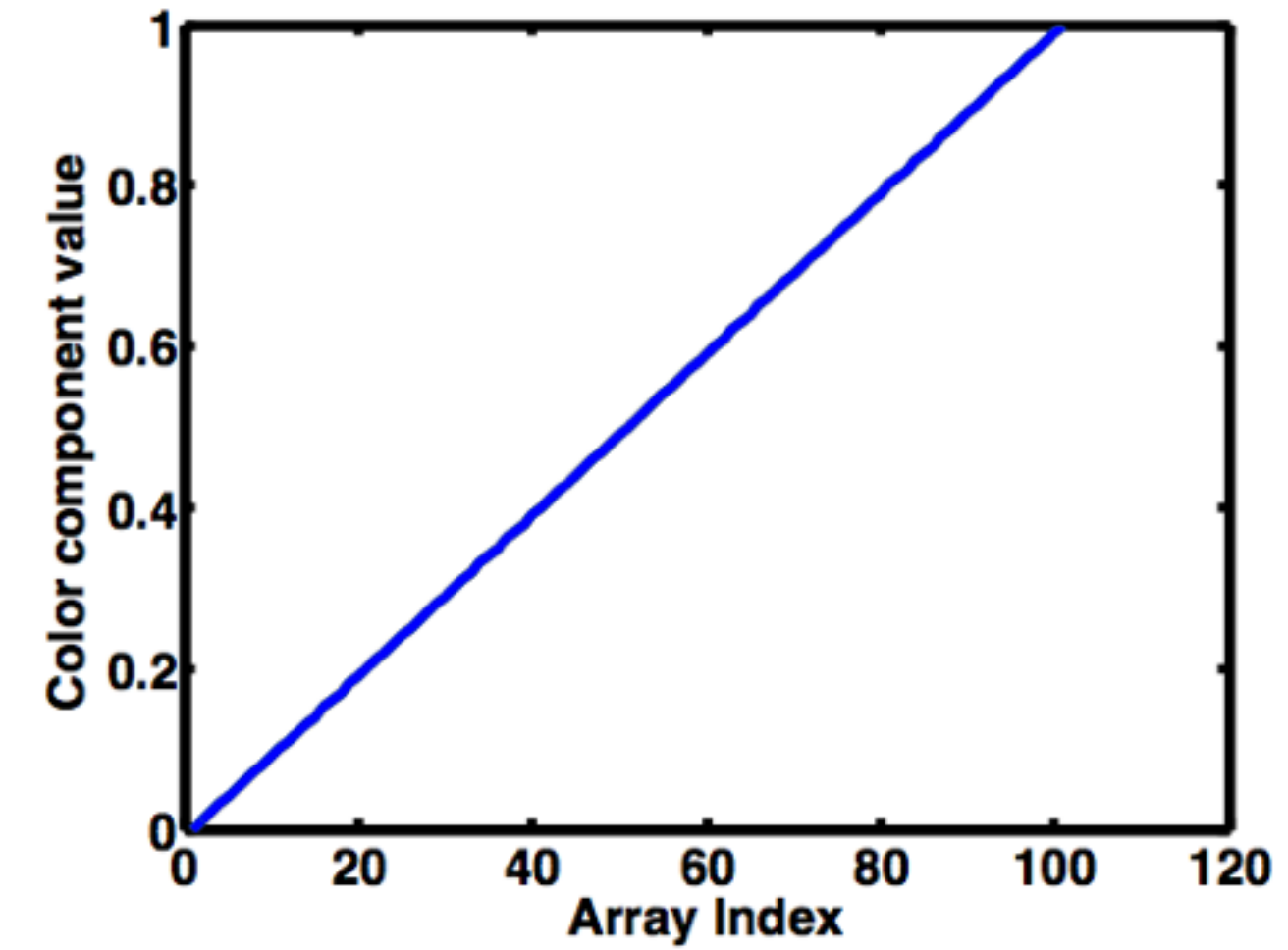
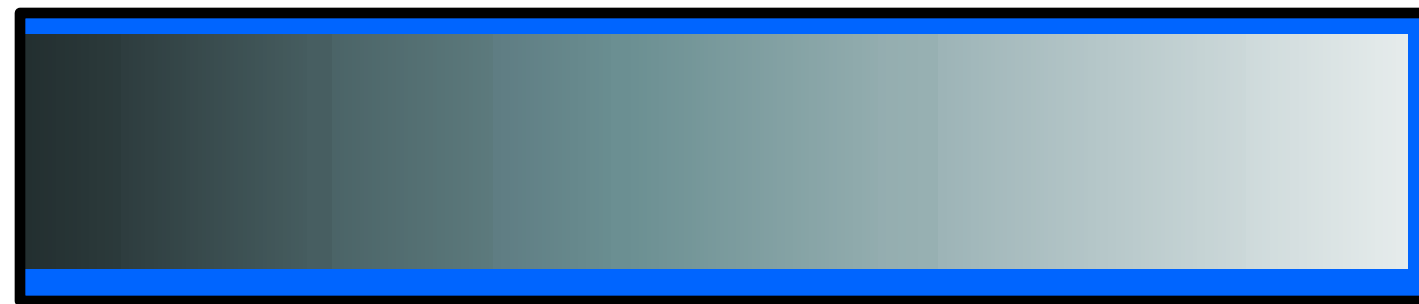
<- Using the built-in 'hot' color map

Using our color map->

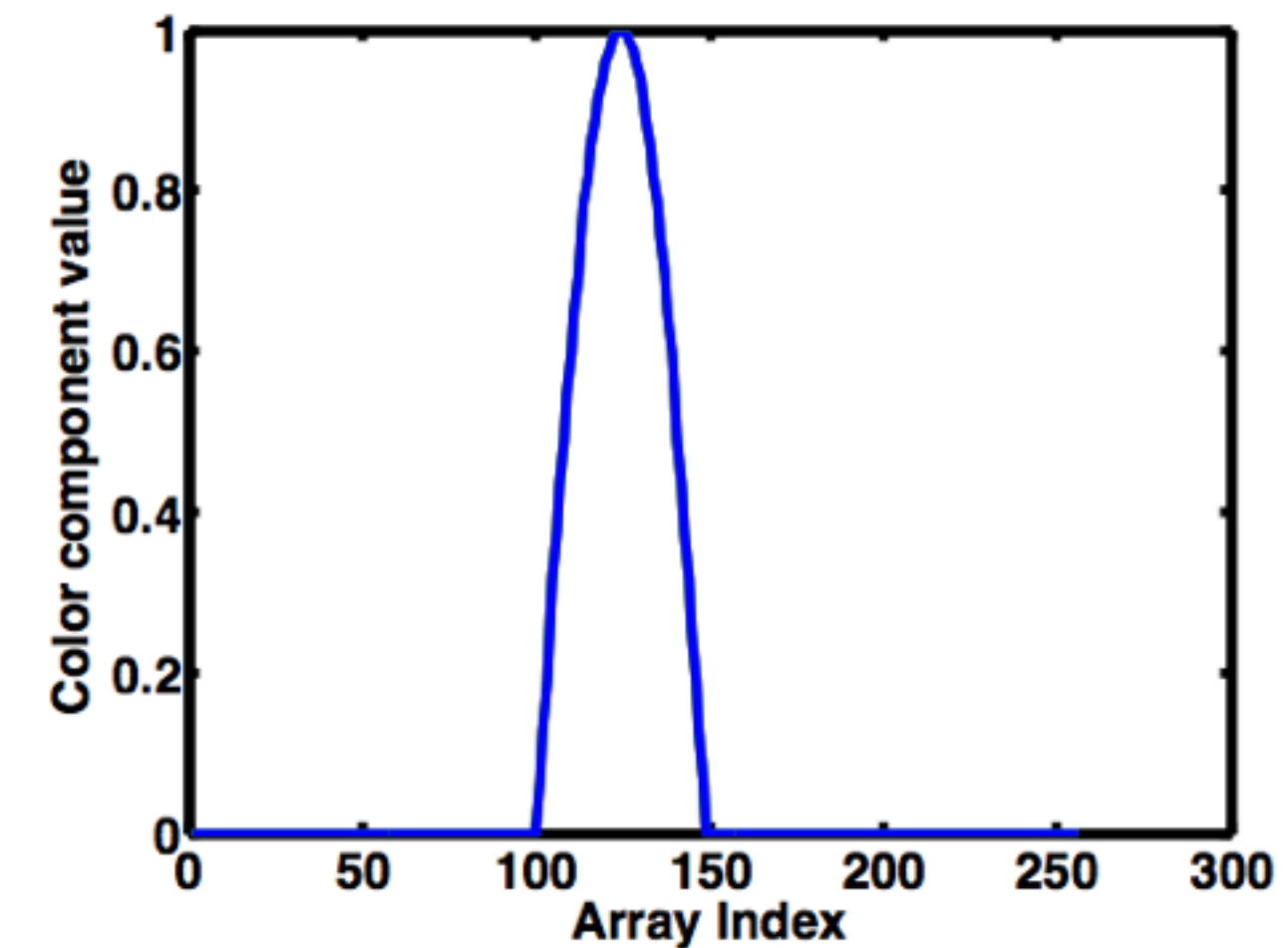
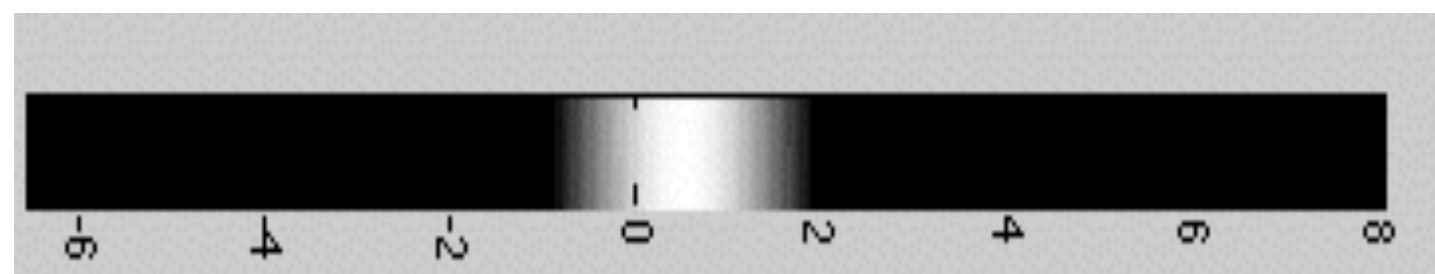


# Other plots vs. custom color maps

- Grayscale?



- Compressed?





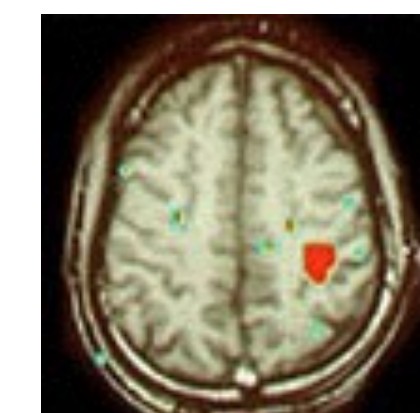
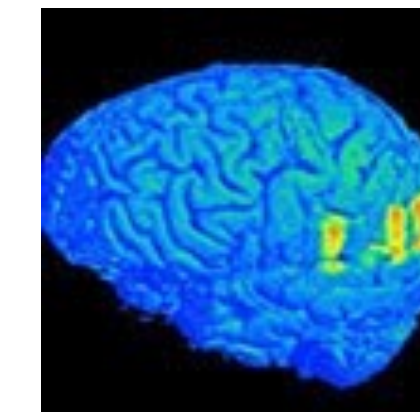
# Python implementation...

- To python...

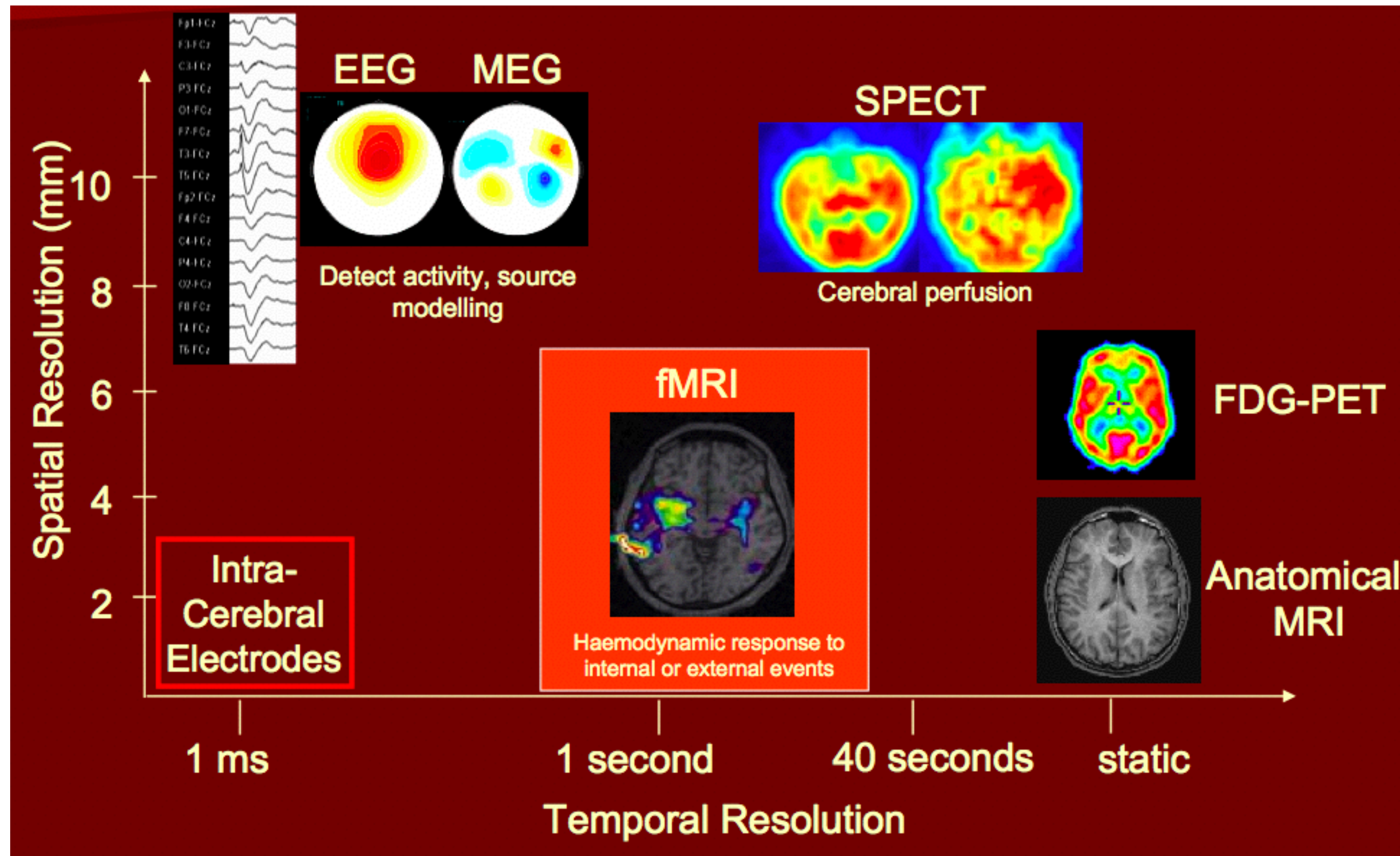


# Consider again an old question

- Suppose I have just performed a brain imaging study where I recorded MRI *and* EEG data of subjects while they performed cognitive tasks *and* they had to perform a motor task (controlling a cursor with a joystick)
- The EEG is sampled at 2kHz (2000 samples/sec)
- The joystick and computer screen updates at 60Hz
- The MRI updates at approximately 1Hz
- How do I analyze this data in light of the fact that there are three very different sample rates?









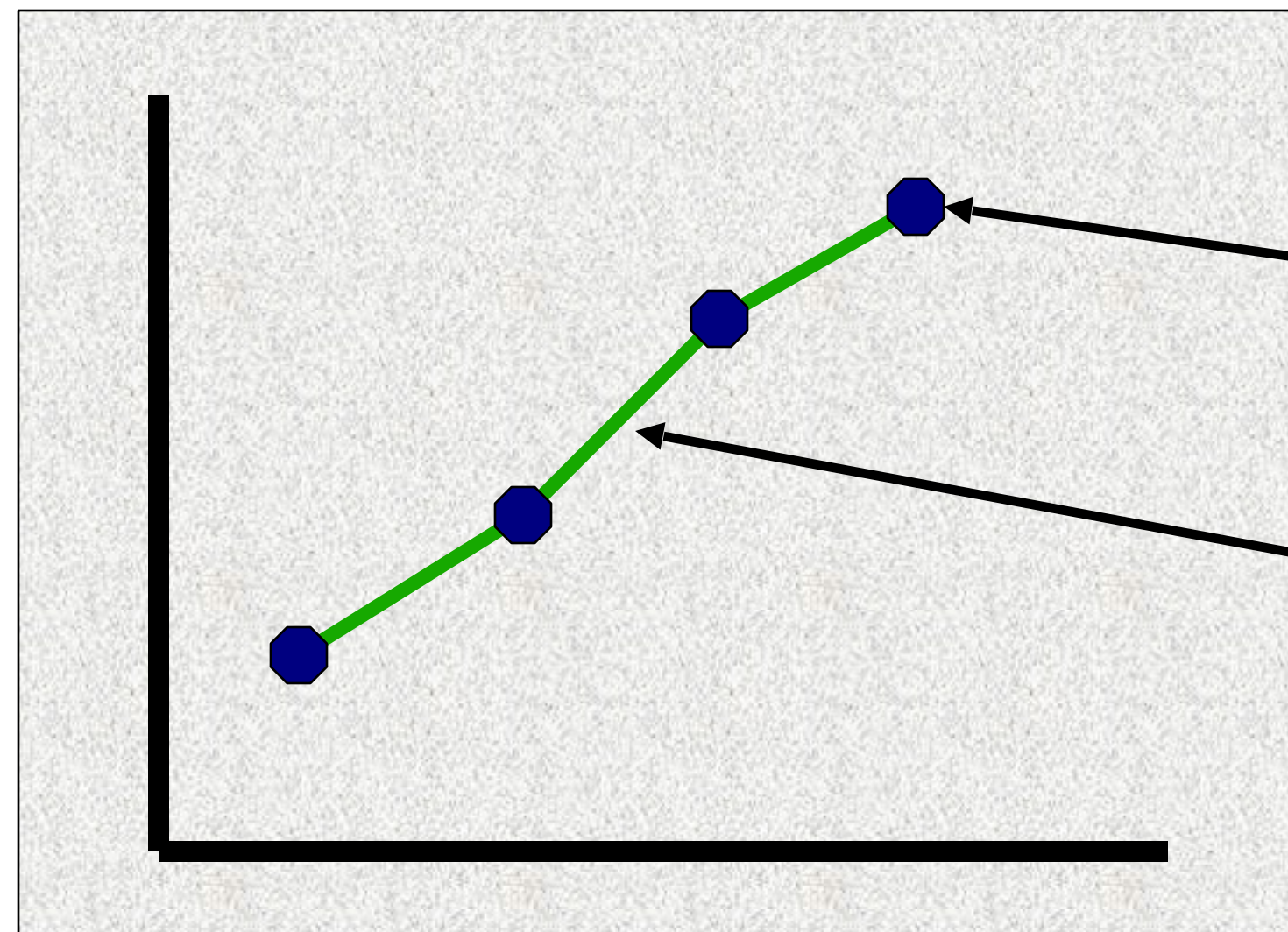


# Answering the question...

- I could *up-sample* the motor data
- But what if I want a smooth curve between points, and I know that the human does not inject significant disturbances between points?
  - We don't want to do least squares because we want something to pass exactly through all data points!
  - I could fit a curve that goes **through** all the data points

# Interpolation defined

- Given a set of data points, we can construct a curve which fits exactly through each datapoint



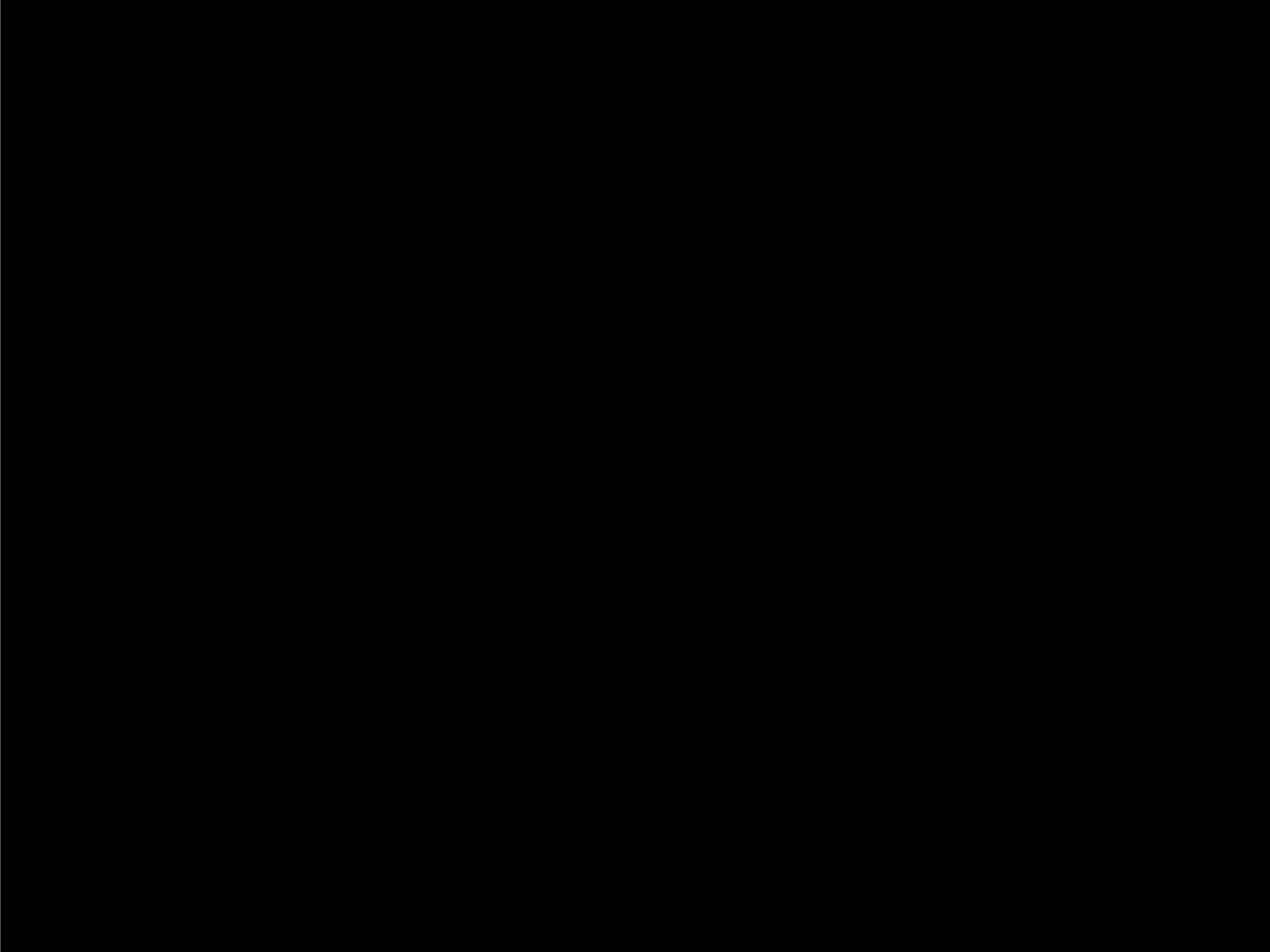
Given this set of datapoints

We want to fit a curve, or piecewise fit curves which pass exactly through each point

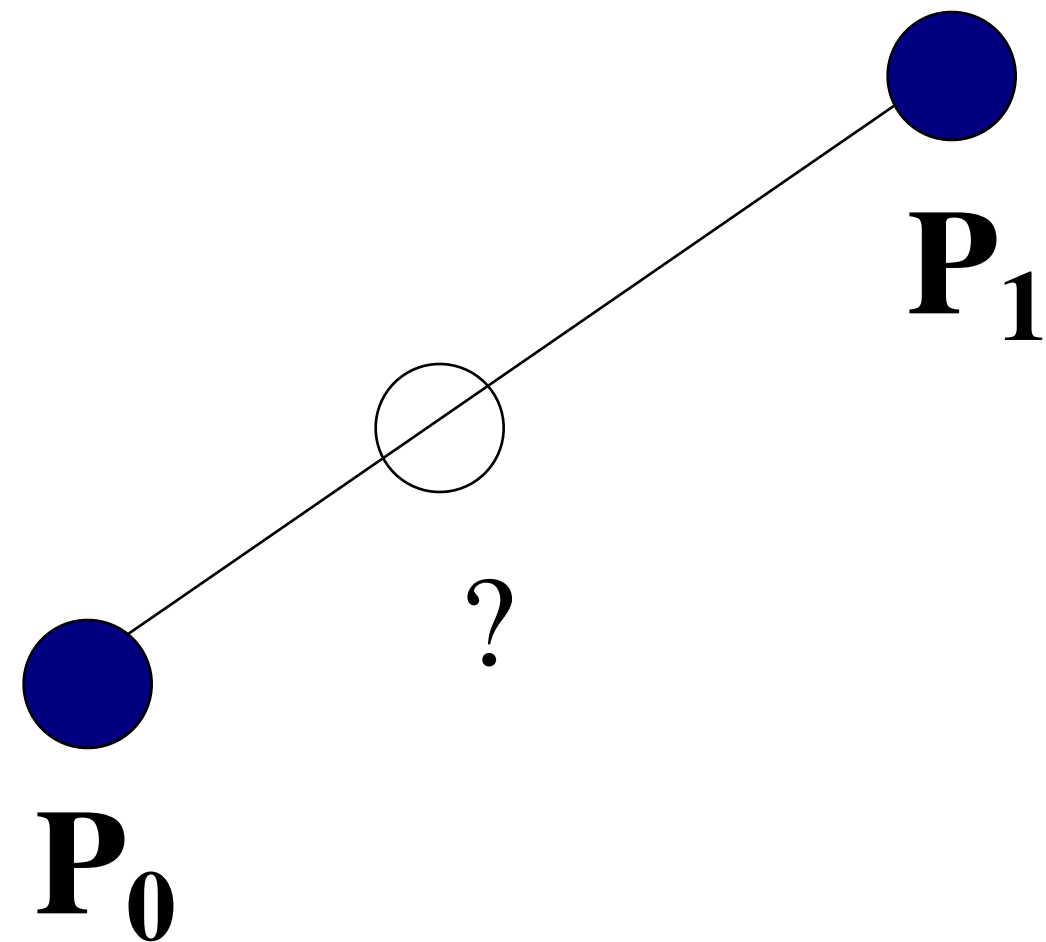


# What can you do with this?

- Match data sampled at different rates
- Create surface plots with varying resolutions (mip-mapping for example with texture mapping)
- Create algorithms which store simplified representations of functions like sine, exponentials, etc (a lookup table), and when faced with points in between the stored values, the algorithm can interpolate between them
  - **Simplifies computation time**
- Create algorithms which take small amounts of data and interpolate to build models in an optimal way to make decisions based on



# Linear interpolation (“LERP”)



We use a parametric curve to blend between the two points:

$$P(t) = (1 - t)P_0 + tP_1$$

Often this is written in the more efficient form:

$$P(t) = P_0 + t(P_1 - P_0)$$

*There are less computations, only compute  $P_1 - P_0$  once per pair of points*

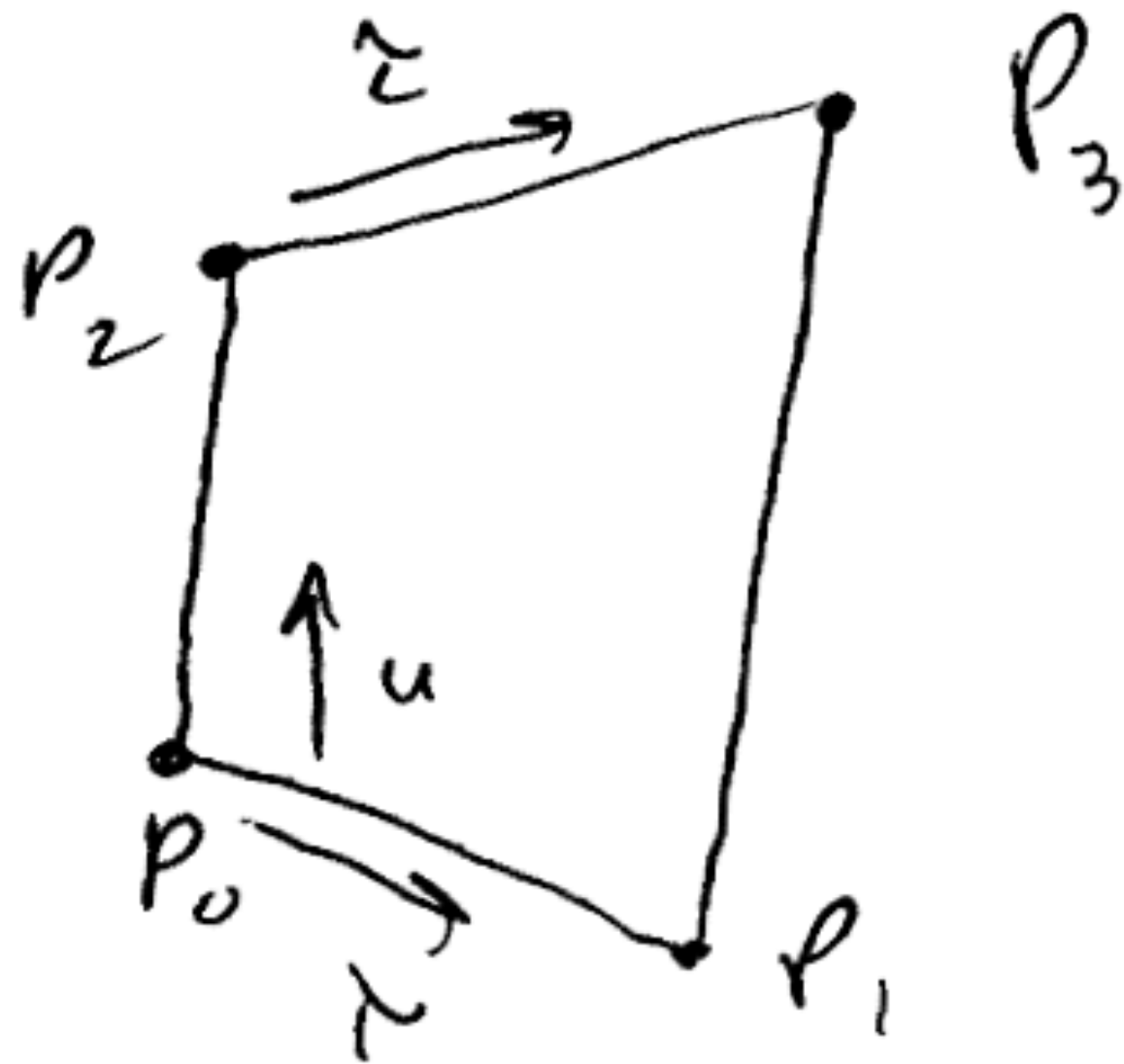
In 3D:

$$x(t) = (1 - t)x_0 + tx_1$$

$$y(t) = (1 - t)y_0 + ty_1$$

$$z(t) = (1 - t)z_0 + tz_1$$

# Bilinear interpolation (“BERP”)



$$P_{01}(t) = (1 - \tau)P_0 + \tau P_1$$

$$P_{23}(t) = (1 - \tau)P_2 + \tau P_3$$

$$P_{0123}(t) = (1 - u)P_{01} + uP_{23}$$

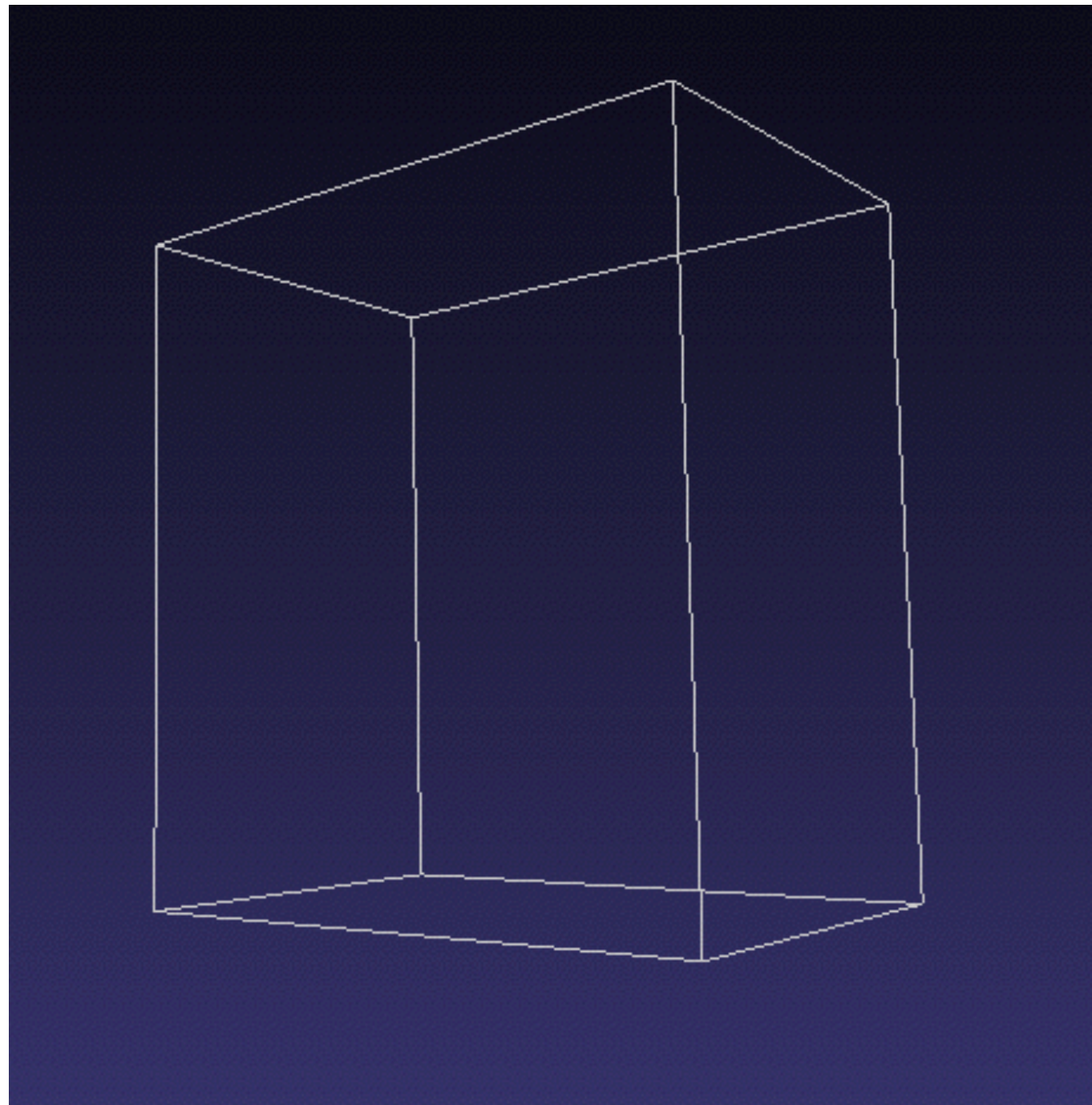
- Substituting the first two into the third:

$$P_{0123}(t) = (1 - \tau)(1 - u)P_0 + \tau(1 - u)P_1 + (1 - \tau)uP_2 + \tau uP_3$$

Thus given 4 points, we can find an interpolated point anywhere in the space between them



# Trilinear interpolation (“TERP”)



- How might you derive this such that we can interpolate to any point inside this cube?
  - **Same as LERP and BERP but a third interpolation parameter (another dimension)**

The points do NOT have to be evenly spaced



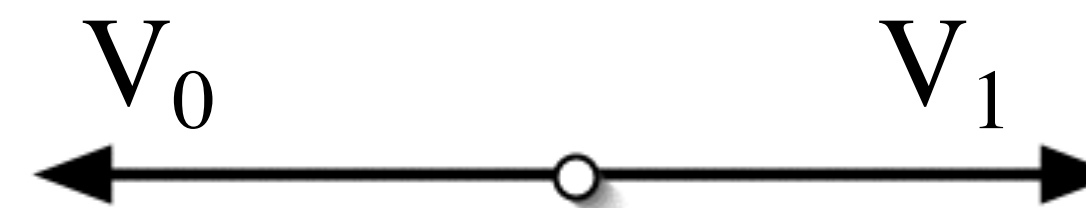
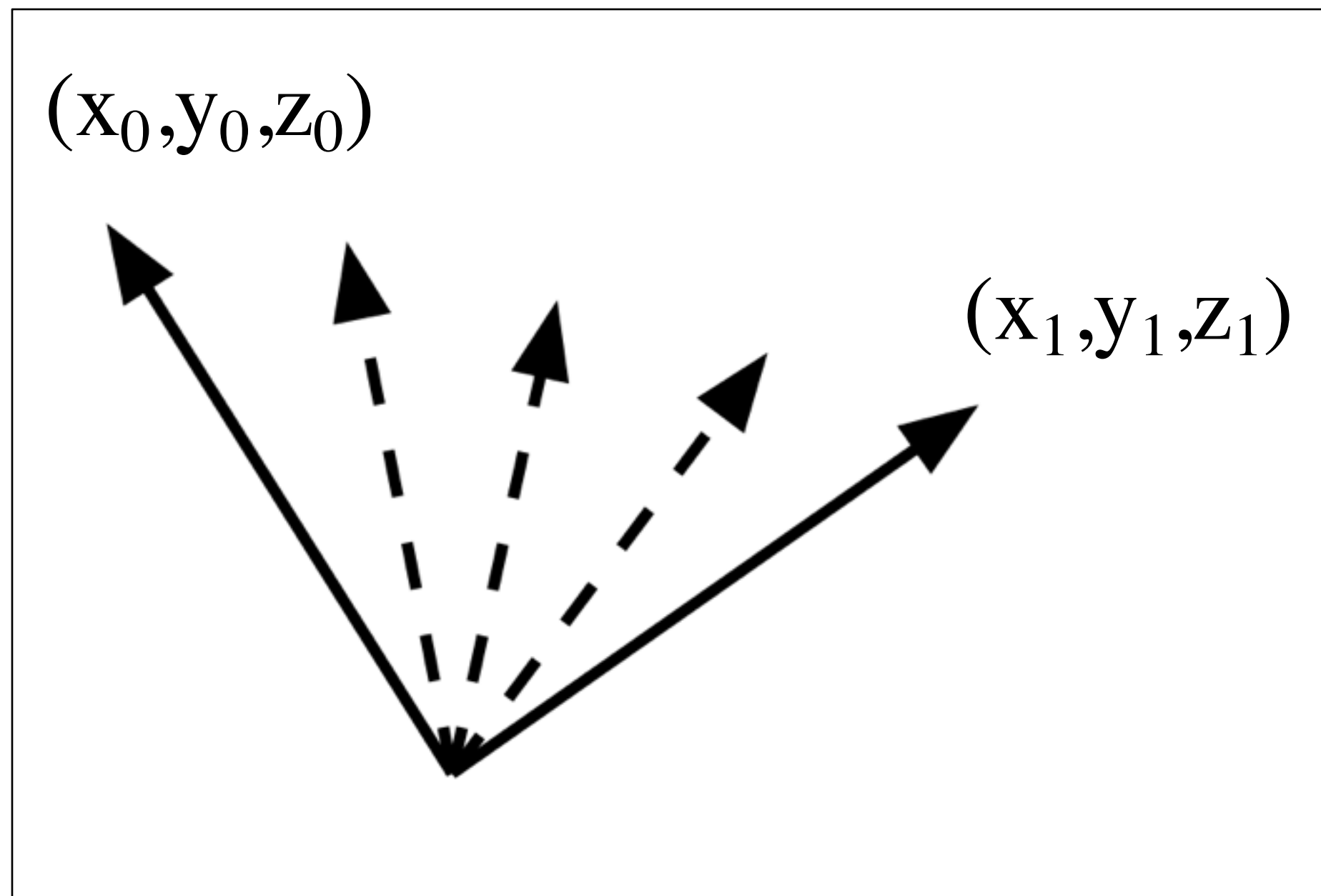
## **An important note about interpolating...**

- By interpolating, you are not truly creating new data, you are blending between existing data
- Use with caution, since significant things might be happening between sample points if your data is too spread apart



# Spherical linear interpolation (“SLERP”)

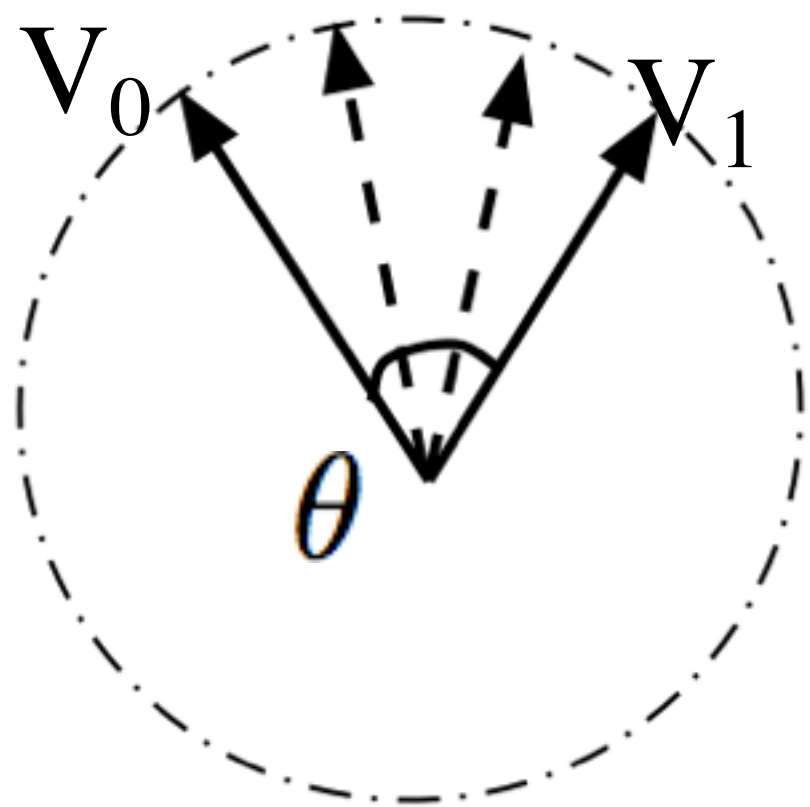
- Let’s say we have two vectors we want to interpolate between:



In case the angle =  $\pi$  (180°) we get a zero length set of vectors by our other interpolation method!

# SLERP continued

- We need a circle in 2D with the center at the origin that passes through both vectors, and we'll interpolate over angle between the vectors, not length of the vectors



Solve for theta by taking inverse cosine of both sides, and noting:

$$\cos^{-1}(\cos(\theta)) = \theta$$

$$\cos(\theta) = \frac{V_0 \cdot V_1}{\|V_0\| \|V_1\|}$$

*This is from the definition of dot products*

$$\cos^{-1}(\cos(\theta)) = \cos^{-1}\left(\frac{V_0 \cdot V_1}{\|V_0\| \|V_1\|}\right)$$

$$\theta = \cos^{-1}\left(\frac{V_0 \cdot V_1}{\|V_0\| \|V_1\|}\right)$$

# SLERP (II)

- Now we can use the same interpolation equation, but with angles...we need to use a sine of the angle we're interpolating, since we're rotating about a circle:

$$\bar{V}(\tau) = \frac{\sin[(1-\tau)\theta]}{\sin(\theta)} \bar{V}_0 + \frac{\sin[\tau\theta]}{\sin(\theta)} \bar{V}_1$$

- Now as Tau goes from 0-1, our vectors are interpolated from  $V_0$  to  $V_1$
- So we compute the angle between the vectors by the dot product equation (with the cosine inverse from the prev. slide)